ESC-TR-2003-075

Project Report
PCA-APP-1

Polymorphous Computing Architecture (PCA) Application Benchmark 1: Three-Dimensional Radar Data Processing

J.M. Lebak
J.S. McMahon
M. Arakawa

19 November 2001
Issued 23 April 2004

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON, MASSACHUSETTS

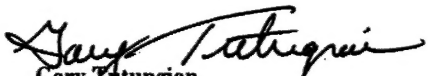
Prepared for the Defense Advanced Research Projects Agency
under Air Force Contract F19628-00-C-0002.

This report is based on studies performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. This work was sponsored by DARPA/ITO under Air Force Contract F19628-00-C-0002.

The ESC Public Affairs Office has reviewed this report, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER



Gary Tatungian
Administrative Contracting Officer
Plans and Programs Directorate
Contracted Support Management

Massachusetts Institue of Technology
Lincoln Laboratory

Polymorphous Computing Architecture (PCA) Application Benchmark 1: Three-Dimensional Radar Data Processing

J.M. Lebak
J.S. McMahon
M. Arakawa
Group 102

Project Report PCA-APP-1

19 November 2001
Issued 23 April 2004

Lexington Massachusetts

ABSTRACT

The DARPA Polymorphous Computing Architecture (PCA) program is building advanced computer architectures that can reorganize their computation and communication structures to achieve better overall application performance. As part of the PCA program, MIT Lincoln Laboratory has been asked to provide examples of defense-oriented applications that will challenge the candidate architectures. This report describes an example airborne radar data processing application. Several example application parameter sets are given that reflect the range of requirements spanned by current systems.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. Introduction

At the PCA Morphware forum meeting in October 2001, the community requested that Lincoln Laboratory develop a benchmark, including performance requirements, distilled from a realistic application. The purpose of the benchmark is to provide realistic performance requirements that will stress the capabilities of PCA hardware and software while remaining simple enough to be easily understandable. Different application parameter sets are provided, each with their own corresponding performance requirements. The intent is to allow the designers to explore *morphing* the system to meet these different sets of requirements.

After surveying several applications, the authors chose to provide an example from airborne radar data processing, as these are among the most challenging problems from a performance standpoint. This particular example involves a three-dimensional data set. Many of the signal processing and data movement kernel functions described in the kernel benchmark report [1] are present in this benchmark. The parameter sets included with this example are chosen to represent actual radar systems and vary widely in order to provide good coverage in application space. The performance requirements are chosen to represent actual radar systems.

The integrated radar-tracker application [2] is in some ways a superset of the functionality described here. That application was designed to provide additional computation requirements to challenge PCAs. Nevertheless, this application is a compact example that can be used to explore the capabilities of PCAs.

This document first describes rough criteria for evaluating the implementation of the benchmark on particular PCA systems. It goes on to describe the benchmark, including the input data set, algorithm details, and performance requirements. Finally, it describes example morphing scenarios for the benchmark.

2. Benchmark Evaluation

There are three important aspects of a PCA system that must be considered in evaluating any implementation of this benchmark: the performance of the benchmark, the application programming interface, and morphability. In this section, we examine each of these aspects in more detail.

2.1 Performance

The first dimension for PCA evaluation is performance. The most important way this application example stresses performance is by providing throughput and latency requirements. The values supplied are intended to be representative not only of current radar performance requirements, but also of those expected in the near future.

The achieved performance on a processing flow is a function of how that flow is mapped to the available computation and communication resources. In a PCA, the resources are not fixed but negotiable. Any intermediate abstractions that are used in the realization of the application on hardware must have several important features. First, the abstractions must insure correctness of execution. Second, the abstractions must have low overhead so they can be manipulated easily and quickly. Third, they must be comprehensive and complex enough to result in efficient use of the hardware, yet simple enough to have low overhead. Finally, the abstractions must be portable between different PCAs.

Since a major purpose of the benchmark is to stress the hardware and software architecture, performance parameters were chosen so that mappings are non-trivial, forcing the efficient and robust use of abstraction at each level of translation. An evaluation of any PCA system must consider the amount of hardware that would be used to meet the requirements and any special aspects of the configuration of the hardware.

2.2 Application Programming Interface

The next dimension for PCA evaluation is the programming interface. The benchmark assists in addressing API issues by forcing the designer to think about the different users of the system and how those users would describe different types of applications to the processor. Issues to consider in API design include expressive power, portability, isolation of hardware details from high-level users, roles for compilers, libraries, and operating systems, and differing paradigms such as object-oriented, functional, etc. In examining the API problem for a specific benchmark example, many of these issues can be illuminated.

In addition to the "standard" API issues, PCA raises another important API issue: the communication and flow of requirements information from the application or mission to the hardware. Since PCAs must react to changing mission requirements, the specification of those requirements must at some level be included in the API. Furthermore, the specification must be able to be given in a scalable way so that the application will not "break" when run on systems with different processing power. The benchmark provided contains requirements information not only to set performance goals, but also to illuminate this API issue. An evaluation of a PCA system must consider the programming methods that would be used to implement the benchmark and how the requirements are described to the PCA system.

2.3 Morphability

The final and distinctive dimension for PCA evaluation is the concept of morphability. The premise of PCA is that hardware architectures will *morph*, or reconfigure, in response to changing application requirements and system conditions. One aspect of morphing is the ease, efficiency, and latency with which machines can morph. Morphing which can be performed *in mission* ("hot morphing") is obviously more challenging that morphing performed *between missions* ("cold morphing"). It should be possible to perform in-mission and between-mission morphing using the same mechanisms and abstractions, making the distinction between these two types of morphing largely a matter of timescale.

A more important distinction is the amount of interaction between the application and the system that morphing requires. Morphing can be regarded as adaptation of the system resources to meet application requirements. We can describe three different morphing scenarios in terms of the system aspect that has changed, as described below.

1. *Application-driven morphing* occurs when the application requirements change but the available system resources remain the same. The system will morph to meet the new requirements. This category includes parameter changes, changes in the amount of workload, and changes in the type of processing (for example, threading versus streaming). It is assumed that the system will be able to meet the new requirements.

2. *System-driven morphing* occurs when the application requirements have not changed but the system resources have changed for some reason that is not application-visible (for example, to meet the processing demands of another application, to recover from a system fault, or for power conservation). The system must reorganize resources and continue to meet the application requirements.

   System-driven morphing can only be performed if the system has some excess capacity over the minimal application requirements. The application will not need to be directly involved in this type of morphing, but it will provide requirements limiting how often this may occur, how long it can take, how much data loss is tolerable, and how much delay is allowed in processing data sets as a result of system-driven morphing.

3. *Cooperative morphing* occurs when there is feedback between the system resources and the application requirements. The application has progressed beyond merely making demands of the system as in category 1: it queries the system for what is possible and scales its demands accordingly. This implies that the PCA system provides a feedback or discovery mechanism.

The minimum level of functionality for PCA is the application-driven morphability, category 1. Category 2 morphing requires some level of resource management in the PCA system itself. Compared to category 2 morphing, category 3 morphing requires only modest additional functionality from the PCA system: most of the burden of interaction must be borne by the application developer in this case.

The benchmark specifies two morphing scenarios, involving application parameter changes (category 1 morphing) and fault tolerance (category 2 morphing). An evaluation of a PCA system must consider the process used to perform the morphing, and how quickly such morphing could be done.

3. Input Data Set

The input data to the application is a three-dimensional data set whose dimensions are *channels*, *range*, and *pulses*. This data set is shown conceptually in Figure 1. A series of N_{pri} pulses are sent by the radar, and for each pulse, a set of N_{rg} range returns is collected on a set of N_{ch} parallel input channels. The output from the sensor is therefore a vector corresponding to a particular value of range and a particular pulse in the sequence.

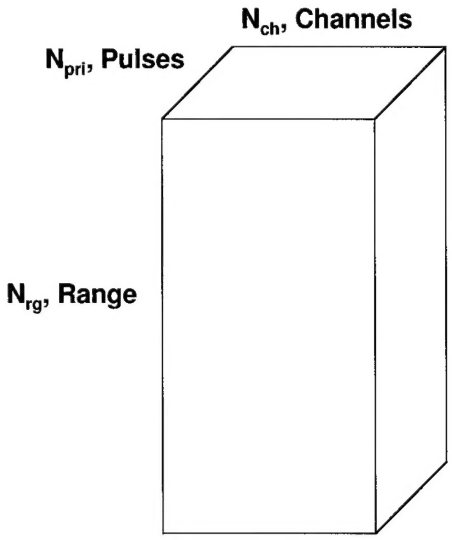


Figure 1. Input data set dimensions.

For any particular pulse in the sequence, a matrix of size $N_{ch} \times N_{rg}$ is collected. Data corresponding to a particular pulse is completed before data corresponding to the next pulse begins. Therefore, range is sometimes referred to as the *fast time* dimension and pulses as the *slow time* dimension. When referring to individual elements of the data cube C, we refer to them as $C(x, y, z)$, where x is the channel index, y is the range index, and z is the pulse index.

Input data arrives as 16-bit integer data in two streams, the 'I' and 'Q' or real and imaginary components of the complex input data. At some point (almost certainly by the beamforming stage), this input data would need to be converted to floating-point. However, the implementation is free to do this conversion either before, after, or during either of the initial filtering stages. The choice of the point of conversion will affect the speed and the accuracy of the implementation.

4. Algorithm

The block diagram of the benchmark processing is given in Figure 2. The application consists of four stages: pulse compression, Doppler filtering, beamforming, and detection. The first two stages perform preliminary processing on the data similar to the low-pass filtering stage of the multi-stage application. The beamforming stage transforms the filtered data to allow detection of signals coming from a particular set of directions of interest, just as in the multi-stage application. The detection stage determines whether targets are actually present in the beamformed data and performs simple grouping and parameter estimation operations on those targets. Each stage is described in more detail in the following sections.

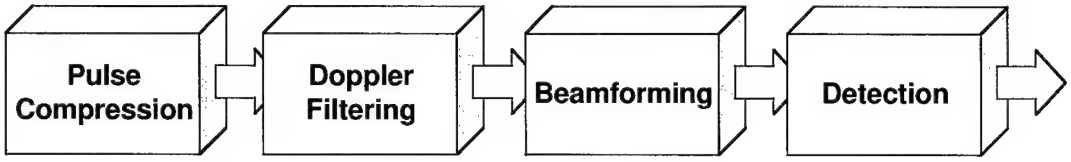


Figure 2. Block diagram of benchmark application.

Most of the processing stages have some algorithm-specific parameters that will vary in different processing scenarios. These parameters are summarized in tables at the end of each processing stage. Table 4 on page 12 summarizes all the data set, algorithm-specific, and performance parameters.

4.1 Pulse Compression Stage

The purpose of the pulse compression stage is to apply a decimating finite-impulse response (FIR) filter operation to the range data for each pulse and channel. The input to each FIR filter operation is a vector x, as shown in Figure 3. In Matlab notation, this vector can be described as $x = C(i, :, k)$, that is, a slice of the data cube containing all the range gates for a particular channel i and pulse k. Additional parameters for the FIR operation are described in Table 1.

The FIR filter has a set of impulse response coefficients $w[l], l \in \{0 \ldots N_{pc} - 1\}$. It also has the option to *decimate*, that is, to reduce the size of the input data used in the operation, by some factor D. If the size of the input data is N_{rg}, the size of the output data is $\lceil N_{rg}/D \rceil$. The output of the filter, y, is the convolution of w with the input x:

$$y[j] = \sum_{l=0}^{N_{pc}-1} x[j * D - l]w[l], \text{for } j = 0, 1, \ldots \left\lceil \frac{N_{rg}}{D} \right\rceil - 1. \quad (1)$$

Although equation (1) gives the mathematical definition of the FIR filter, the most efficient implementation of the filter may not use this formula directly. If the length of the filter N_{pc} is short and

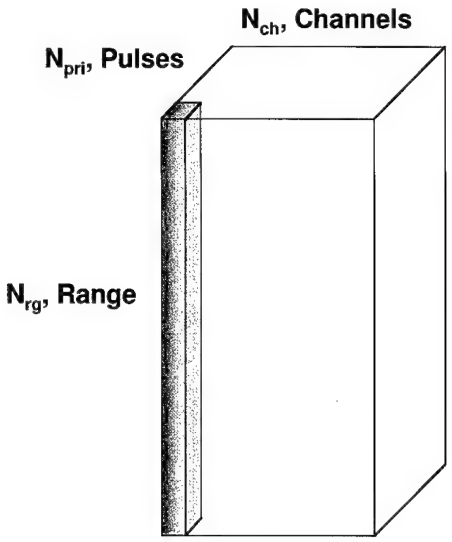


Figure 3. Input data for the pulse compression stage. The shaded area shows the data operated on by each FIR filter operation. This operation is performed for each channel and each pulse.

the data size N_{rg} is large, then an actual convolution might be explicitly implemented. Otherwise, a fast convolution using FFTs might be performed, possibly even considering an overlap-and-save method [3].

Table 1.

Pulse Compression Algorithm Parameters.

Name	Description
N_{pc}	Number of taps in the pulse compression filter.
D	Decimation factor for pulse compression filter.

4.2 Doppler Filtering Stage

The Doppler filtering stage is a straightforward FFT applied to the pulse data for each range gate and channel. The specific algorithm for implementing the FFT is not specified. The length of the FFT is assumed to be equal to the number of pulses. The input data for each Doppler filtering operation is shown in Figure 4: it is a vector that can be described as $C(i, j, :)$.

There are no algorithm parameters specific to this stage. After the Doppler filtering stage, the pulse dimension of the data cube is referred to as the *Doppler* dimension.

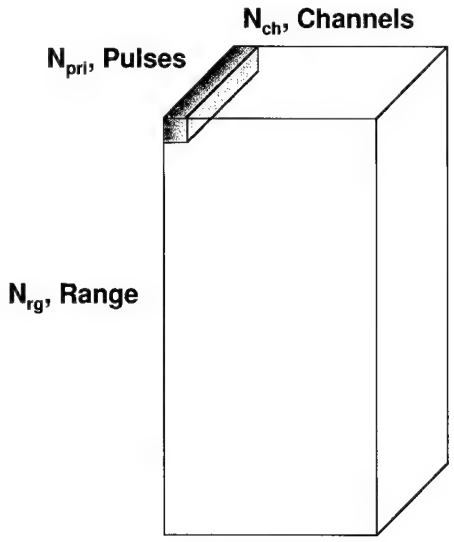


Figure 4. Input data for the Doppler filtering stage. The shaded area shows the data operated on by each FFT operation. This operation is performed for each channel and each range gate.

4.3 Beamforming Stage

In the beamforming stage, the filtered data matrix corresponding to each Doppler is transformed into a set of beams. Each beam represents the array output for a particular direction. The beam matrix Y is formed by multiplying the filtered input X by a beamforming matrix W,

$$Y = W^H X. \tag{2}$$

The matrix X is size $N_{ch} \times N_{rg}$, and W is size $N_{ch} \times N_{bm}$, where there are N_{bm} beams formed from the N_{ch} channels and N_{bm} is described in Table 2. An example beamforming matrix W can be computed from two vectors a and b using the equation

$$W^H = e^{j(ab^H)}. \tag{3}$$

Vector a is a simple ramp function:

$$a[i] = (i-1), i = 0, 1, \ldots, N_{ch} - 1. \tag{4}$$

Vector b is a *phase ramp* function:

$$b[i] = 2\pi K \sin\left(-\frac{\pi}{4} + \frac{i\pi}{2(N_{bm}-1)}\right), i = 0, 1, \ldots, N_{bm} - 1, \tag{5}$$

where K is a constant related to the wavelength of the radar.

Figure 5 illustrates X, the cross-section of the data cube that is operated on by the beamforming stage. The matrix represents all the range and channel information for a particular Doppler: it can

be described in Matlab notation as $X = C(:, :, k)$. Note specifically the implications that this has for data movement. In the previous stage, the Doppler filtering operations were performed in the direction of the pulse dimension of the data cube. In this stage, all the data corresponding to a particular Doppler must be used to perform the matrix multiply described by equation (2). This requires that a rearrangement of the data cube, similar to the *corner-turn* kernel benchmark [1], has to be performed. This is *not* to imply that the matrix multiplication described in equation (2) must be performed on a single processor: it may be parallelized in range. As in the previous set of API examples, this particular detail should be transparent to the application programmer.

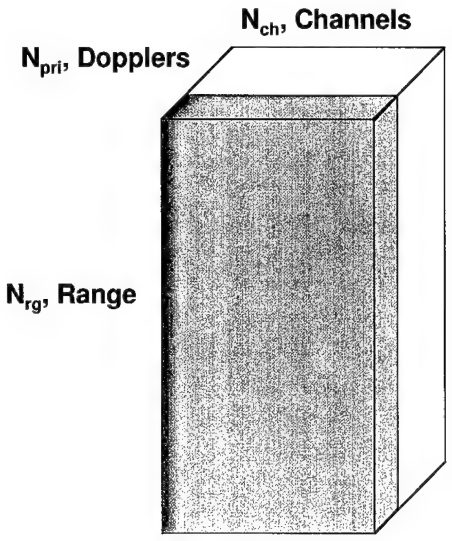


Figure 5. Input data for the beamforming stage. The shaded area shows the data operated on by each matrix multiply operation. This operation is performed for each pulse.

Table 2.

Beamforming Algorithm Parameters.

Name	Description
N_{bm}	Number of beams formed.

4.4 Detection Stage

The detection stage adds processing that is data-dependent, compared to the matched filtering operation shown in the original multi-stage application. This stage consists of three sub-steps, referred to as constant false-alarm rate detection (CFAR), three-dimensional grouping, and centroiding. Roughly speaking, these three steps can be described as finding targets, eliminating multiple detections associated with the same target, and estimating the true position of a target.

These three steps are described in more detail in the following sections. See Table 3 for definitions of parameters used in this stage.

4.4.1 CFAR detection

During CFAR detection, a local noise estimate is computed from the $2N_{cfar}$ range gates near the cell $C(i,j,k)$ under test. A number of guard gates G immediately next to the cell under test will not be included in the local noise estimate (this number does not affect the throughput). For each cell $C(i,j,k)$, the value of the noise estimate $T(i,j,k)$ is calculated as

$$T(i,j,k) = \frac{1}{2N_{cfar}} \sum_{l=G+1}^{G+N_{cfar}} |C(i,j+l,k)|^2 + |C(i,j-l,k)|^2. \tag{6}$$

The range cells involved in calculating the noise estimate for a particular Doppler bin and beam are shown in Figure 6. For each cell $C(i,j,k)$, the quantity $|C(i,j,k)|^2/T(i,j,k)$ is calculated: this represents the normalized power in the cell under test. If this normalized power exceeds a threshold μ, the cell is considered to contain a target.

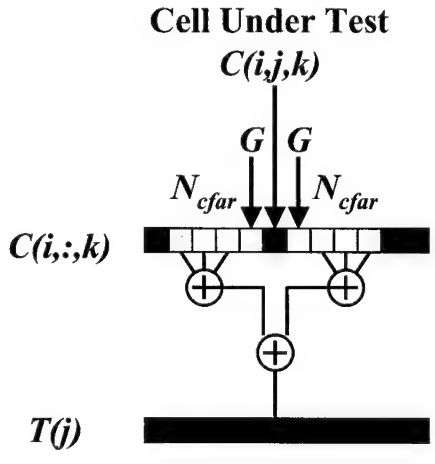


Figure 6. Sliding window in CFAR detection. The example shows the number of guard cells $G = 1$ and the number of cells used in computing the estimate $N_{cfar} = 3$.

The only cells in the data cube that are processed by the following stages are those where a target is detected.

4.4.2 Three-dimensional grouping

During this step, detections in adjacent cells are grouped to avoid having multiple detections associated with the same target. The power of each target detected by the CFAR algorithm will

be compared to the power of each cell in a $3 \times 3 \times 3$ cube centered on that cell. Therefore, each cell under test will be compared to 26 other cells ($27 - 1$, where $27 = 3^3$ and 1 is the cell under test, right in the middle of the cube). If the detection's raw power is the local maximum, it is retained. Otherwise, the detection is discarded; it is considered grouped with one of the higher-power detections in the surrounding cells.

4.4.3 Centroiding

During this step, the location of each specific detection that is reported by the three-dimensional grouping stage will be refined by performing power-weighted centroiding. This operation is performed independently in each of the beam, range, and Doppler bin dimensions in order. The power of the immediately neighboring cells will be considered in the centroiding operation. For example, in the beam centroiding operation on element $C(i, j, k)$, the beam estimate $B(j, k)$ is given as

$$B(j,k) = \frac{\sum_{l=-1}^{1} (i+l) \times |C(i+l,j,k)|^2}{\sum_{l=-1}^{1} |C(i+l,j,k)|^2}. \tag{7}$$

Similar operations are performed in the range and Doppler bin dimensions.

Table 3.

Detection Algorithm Parameters.

Name	Description
N_{cfar}	Number of range gates used in forming the noise estimate from *each* side of the cell under test.
G	Number of guard cells on each side of the cell under test.

5. Performance Requirements

Performance requirements for a typical application can be expressed in terms of *latency*, defined as the number of data collection intervals required to complete the processing of a single data set, and *throughput*, defined as the number of data samples processed in some time interval. For a PCA system, we also define a *maximum data loss* allowable due to a fault.

The throughput for this application is expressed by the *sampling rate* R_s, the rate at which data samples come into the processor, and the *pulse repetition frequency* or PRF, the frequency at which pulses are sent out by the radar.[1] The processor must be able to keep up with both the rate at which samples are received from an individual pulse and with the number of pulses being received.

The latency for the algorithm is expressed in terms of coherent processing intervals (CPIs), which correspond to the amount of time taken to collect data for a particular set of N_{pri} pulses. The CPI time is the product of the pulse repetition interval (the inverse of the pulse repetition frequency) and the number of pulses.

For fault tolerance, we specify that the system must be able to reconfigure in the event of a fault without losing more than a certain number of CPIs of data. We call this number the maximum data loss, q.

[1] Because the radar cannot transmit and receive simultaneously, there is some time during each pulse repetition interval that no samples are taken. Therefore, the product of the number of range gates in a pulse repetition interval and the pulse repetition frequency will not equal the sampling rate.

6. Parameter Sets

Table 4 presents the parameter values for three radar scenarios. The parameters are grouped into three categories: input data set parameters, algorithm parameters, and performance parameters.

Table 4.

Parameter Set Values

Type	Name	Description	Values			Units
			Set 1	Set 2	Set 3	
Input Data	N_{ch}	Number of channels.	48	48	16	channels
	N_{rg}	Number of range gates.	3500	1909	9900	samples
	N_{pri}	Number of pulses.	128	64	16	pulses
Algorithm	N_{pc}	Number of taps in the pulse compression filter.	667	364	100	taps
	D	Decimation factor for pulse compression filter.	1	1	2	
	N_{bm}	Number of beams formed.	20	20	5	beams
	N_{cfar}	Number of range gates used in forming the noise estimate from *each* side of the cell under test.	10	10	20	cells
	G	Number of guard cells on each side of the cell under test.	2	2	4	cells
Performance	R_s	Sampling rate.	1	1	10	MHz
	PRF	Pulse repetition frequency.	240	440	1000	Hz
	CPI	Coherent processing interval.	533	145	16	ms
	L	Latency.	6	6	6	CPIs
	q	Maximum data loss	3	3	3	CPIs

To assist in judging the application workload, we include estimates of the number of floating-point operations in each stage for each parameter set in Table 5.[1] The operation counts assume that the data set is converted to floating-point before the first stage. In the detection stage, the operation counts assume that 5% of all cells contain a target and that 50% of all cells survive target grouping.

[1]Obviously, these counts are not exact, as the operation count will vary based on the exact implementation, when floating-point conversion is performed, etc.

Table 5.

Parameter Set Values

Stage	Operation Counts			Units
	Set 1	Set 2	Set 3	
Pulse Compression	6.85	1.59	0.61	Gflop
Doppler Filtering	0.75	0.18	0.03	Gflop
Beamforming	3.44	0.94	0.05	Gflop
Detection				
CFAR Detection	98.6	26.9	4.4	Mflop
Grouping	11.7	3.2	0.52	$\times 10^6$ compares
Centroiding	8.1	2.2	0.36	Mflop

7. Conclusions

The intent of this application benchmark is to allow designers of PCA systems to explore the implementation of a reasonably-sized application with realistic parameter sets on their architectures. The following criteria are suggested for evaluating these implementations.

Mapping. Mapping is the process of assigning tasks to computing engines and defining all necessary data movement between components. An important item for evaluating the system is a description of the decomposition and mapping of benchmark functionality onto hardware resources. The hardware resources utilized in aggregate must be capable of meeting performance requirements as specified in the benchmark: therefore, an implementation description should include an analysis of how the mapping meets performance requirements. In current systems, such analyses include estimates of the following performance metrics:

- Achieved throughput in Mflop/s or op/s
- Estimated latency in seconds
- Processing efficiency

Processing efficiency is defined as the ratio of achieved computational throughput to peak throughput provided by the allocated hardware resources. Processing efficiency affects both form-factor and system cost and is therefore an extremely important metric for DoD applications.

Programming. Another important consideration for evaluation is a specification of the program that implements the benchmark using the language and programming paradigm envisioned for the particular architecture under study. Such a specification would include a discussion of the software tools – libraries, compilers, the operating system, etc. – that would be used to create the application program, and what roles each would play. Especially important would be a discussion of how the morph state of the hardware is controlled and what abstractions are provided to isolate the programmer from the hardware.

With a description of the implementation focused on these criteria, it should be possible to compare PCA systems to current systems to evaluate their capabilities and potential.

REFERENCES

1. J. Lebak, A. Reuther, and E. Wong, "Polymorphous Computing Architecture (PCA) Kernel-Level Benchmarks," MIT Lincoln Laboratory, Lexington, Mass., Project Report PCA-KERNEL-1, 24 January 2004.

2. J. M. Lebak, "Preliminary Design Review: PCA Integrated Radar-Tracker Application," MIT Lincoln Laboratory, Lexington, Mass., Project Report PCA-IRT-1, 9 April 2002, issued 6 February 2004.

3. Alan V. Oppenheim and Ronald W. Schafer, *Discrete-time signal processing,* Prentice-Hall, Inc., 1989.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE 19 November 2001	3. REPORT TYPE AND DATES COVERED Project Report

4. TITLE AND SUBTITLE

Polymorphous Computing Architecture (PCA) Application Benchmark 1: Three-Dimensional Radar Data Processing

5. FUNDING NUMBERS

C — F19628-00-C-0002

6. AUTHOR(S)

J.M. Lebak, J.S. McMahon, and M. Arakawa

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Lincoln Laboratory, MIT
244 Wood Street
Lexington, MA 02420-9108

8. PERFORMING ORGANIZATION REPORT NUMBER

PR-PCA-APP-1

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

DARPA/ITO
3701 Fairfax Drive
Arlington, VA 22203-1714

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

ESC-TR-2003-075

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

The DARPA Polymorphous Computing Architecture (PCA) program is building advanced computer architectures that can reorganize their computation and communication structures to achieve better overall application performance. As part of the PCA program, MIT Lincoln Laboratory has been asked to provide examples of defense-oriented applications that will challenge the candidate architectures. This report describes an example airborne radar data processing application. Several example application parameter sets are given that reflect the range of requirements spanned by current systems.

14. SUBJECT TERMS

15. NUMBER OF PAGES
20

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Same as Report	Same as Report	Same as Report